Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Andraž Omahen

Simulacija prometnih tokov

diplomsko delo

visokošolski strokovni študijski program prve stopnje računalništvo in informatika

Mentor: višji pred. dr. Aleksander Sadikov
Somentor: doc. dr. Gregor Papa

Ljubljana 2015

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Cilj diplomske naloge je izpopolniti simulator za cestno omrežje s semaforiziranimi križišči in simulirati vožnjo vozil s ptičje perspektive. Možno naj bo opazovati kje in kako se promet zgoščuje. Vsako križišče ima lahko različno nastavljene semaforje. Kandidat naj simulator pripravi tako, da se bo lahko vzporedno izvajalo več instanc le-tega, pri čemer bo vsaka instanca imela različne nastavitve semaforjev. Program naj vsebuje tudi grafični vmesnik za nastavljanje semaforjev, dodajanje in spreminjanje cest ter križišč. Kandidat naj za demonstracijo simulatorja zgradi omrežje znotraj ljubljanske obvoznice kjer naj ceste povezujejo osnovna enopasovna križišča.

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Andraž Omahen sem avtor diplomskega dela z naslovom:

Simulacija prometnih tokov

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom višj. pred. dr. Aleksandra Sadikova in somentorstvom doc. dr. Gregorja Pape,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela na svetovnem spletu preko univerzitetnega spletnega arhiva.

V Ljubljani, dne 15. marec 2015 Podpis avtorja:

Na tem mestu bi se rad zahvalil osebju Inštituta »Jožef Stefan«, predvsem doc. dr. Gregorju Papi, dr. Vidi Vukašinović in izr. prof. dr. Petru Koroščcu. Posebna zahvala gre tudi mojemu mentorju višj. pred. dr. Aleksandru Sadikovu. Zahvalil bi se tudi dr. Martinu Treiberju, ki nam je dovolil, da za pričujočo diplomsko nalogo uporabimo njegov simulator ter Teu Kukuljanu za pomoč pri generiranju križišč. Zahvaljujem se tudi družini in prijateljem za veliko podporo.

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
IDM	Intelligent driver model	Inteligentni model voznika
MOBIL	Minimizing Overall Braking decelerations Induced by Lane changes	Zmanjšanje vseh zavornih pojemkov ki jih povzroča menjava pasov
IDE	Integrated development environment	Integrirano razvojno okolje
JAXB	Java Architecture for XML Binding	Zavezujoča javanska arhitektura XML
XML	EXtensible Markup Language	Razširljiv označevalni jezik
JDK	Java Development Kit	Java razvojni komplet

Povzetek

S simulacijami lahko posnemamo različne procese. Zaradi skokovitega tehnološkega razvoja in vedno večjih računskih zmožnosti jih lahko zaganjamo tudi na osebnih računalnikih. Uporabljamo jih za virtualno ponazarjanje situacij, pri dolgotrajnih procesih pa jih uporabljamo tudi za ponazarjanje pospešenih razmer. Tempo našega življenja vsekakor vpliva na človeka, ki se zato vedno bolj zaveda pomembnosti in vrednosti časa. Veliko ga izgubimo tudi z vožnjo po cestah, ko se nepričakovano znajdemo v prometnem zastoju. Dandanes ima vsak drugi državljan registrirano motorno vozilo, zaradi česar prihaja na cestah do pogostih zastojev.

V diplomskem delu je predstavljeno delovanje programa, ki simulira tok vozil v prometnem omrežju, povezanim s semaforiziranimi križišči. Vozila se obnašajo po modelu IDM. Program smo nadgradili tako, da ga lahko v danem omrežju uporabimo za optimizacijo prometnega toka. Ker je velika problematika tudi zapora cest ali njihova preureditev, smo dodali tudi grafični vmesnik, s katerim lahko uporabnik, če je treba, sestavi ali spremeni cestno omrežje in opazuje posledične spremembe v prometu.

Cilj diplomske naloge je izdelati simulator, v katerem bo mogoče sestaviti semaforizirano cestno omrežje. S takim omrežjem bo mogoče optimirati časovne intervale semaforjev, kar bo zagotavljalo boljši pretok prometa. Glavni del diplomske naloge je pripraviti program za sestavo cestnih povezav, po katerih se vozijo osebna in druga vozila. Poleg povezav imamo tudi križišča, ki so semaforizirana, semaforji pa delujejo z različnimi časovnimi intervali. Program je treba pripraviti tako, da se lahko vzporedno izvaja

več simulatorjev z različnimi nastavitvami časovnih intervalov semaforjev. Cilj tega je omogočiti iskanje in določitev optimalne nastavitve intervalov semaforjev za dano cestno omrežje.

Ključne besede: simulator, prometni tok, optimizacija, semaforizacija, promet.

Abstract

Simulations allow imitating various processes. Due to the growing technological development and increasing computational capabilities of personal computers, simulations can be run on personal computers, too. We use simulations to imitate real-life situations virtually, and, in case of long-lasting processes, we use them to virtually speed-up the process. The pace of our life certainly affects humans, who are increasingly aware of the importance and the value of time. Much of our time is spent on the roads, where unexpectedly we find ourselves in traffic jams. Today, in our country, every second citizen has a registered motor vehicle, which cause congestions and traffic peaks on the road.

This diploma thesis presents the behavior of program, which uses the traffic light enhanced transport network to simulate the traffic flow of vehicles that behave according to the IDM model. We upgraded the program to allow, for the given network, to optimize the traffic flow. Because of the overwhelming problem of road closures or their rearrangement, we added a graphical user interface with which a user can create or modify the road network to observe changes in traffic.

The objective of this diploma thesis is to create a simulator with which it will be possible to construct a traffic lights road network. Using this network, it will be able to optimize the time intervals of traffic lights to improve traffic flow. The main part of this diploma thesis is to prepare a program for the composition of road links, where cars and other vehicles drive through. In addition, traffic lights at junctions are changing at different

time intervals. The program has to be adapted to allow parallel execution of multiple simulators with different traffic light interval settings. The aim is to find the optimal settings of the intervals of traffic lights, the result being the most effective traffic flow.

Keywords: simulation, traffic flow, optimization, traffic lightning, traffic.

Poglavje 1

Uvod

V diplomski nalogi želimo pomagati rešiti problem zgoščevanja prometa in prometnih konic, ki nastajajo ob določenih urah, ko se ljudje odpravljajo na delo ali odhajajo iz služb. Velik problem so zastoji, do katerih prihaja na cestah. Z diplomskim delom želimo optimizirati delovanje semaforiziranih križišč tako, da bo pretok prometa čim boljši, da torej ne bo prihajalo do zastojev na cestah.

Cilj diplomske naloge je sestaviti cestno omrežje, s katerim bi simulirali vožnjo vozil. Vozila se premikajo po modelu IDM[2], ob tem pa lahko opazujemo, kje in kako se promet zgoščuje. Kot izhodišče smo vzeli simulacijsko okolje MovSim[6] nato pa je bilo treba preučiti dele programa: kako deluje in kako je sestavljen, saj temelji na tehnologiji Maven, in sicer v jeziku Java. Maven je odličen projekt izdelave(ang. project build) in orodje za upravljanje od Apache programske skupnosti[4]. Sestaviti je bilo treba omrežje cest, ki so med seboj povezane in imajo semaforizirana križišča. Vsako križišče ima lahko različno nastavljene intervale semaforjev za boljšo prepustnost, zato se simulacija zaporedno večkrat požene z različnimi nastavitvami semaforjev, saj le tako lahko najdemo optimalno rešitev za pretočnost cest. Simulator je že imel implementiran model IDM in MOBIL, namenjen vožnji in premikanju vozil. Imel je tudi narejene scene za vključevanje na vozni pas, izhod iz voznega pasu, hkrati pa je že imel dodane funkcionalnosti kot so semafor in

omejitev hitrosti. Vse od naštetega je znal tudi izrisati.

Moj glavni prispevek k delu so dodatne funkcionalnosti simulatorja, ki smo jih razvili kot sestavni del programa:

- Program smo najprej prilagodili tako, da se lahko zaporedno izvaja več instanc simulatorja hkrati. To nam je uspelo tako, da smo vse splošne statične objekte spremenili v samostojne. Nobena instanca simulatorja si torej ni delila podatkov z drugimi, tako da so vse delovale neodvisno.

- Poleg sestave cestnega omrežja in križišč smo dodali tudi grafični vmesnik za popravljanje in dodajanje cest, tako ravnih kot krivuljastih.

- Dodana pa je bila tudi funkcionalnost iskanja najoptimalnejše poti glede na izbrani kriterij (ali je pomembnejša dolžina poti ali zgoščenost prometa).

- Na ceste lahko postavimo tudi cestne zapore, kar pomeni, da vozila tam ne morejo voziti, ali zaradi nesreče ali dela na cesti.

- V simulator lahko dodajamo tudi različne tipe križišč.

- Za sestavo omrežja smo zgradili osnovno križišče, ki vključuje že implementirane semaforje, vključili pa smo tudi kritične dele cest, da se vozila med seboj ne zaletavajo.

- Iz osnovnih križišč smo sestavili omrežje znotraj ljubljanske obvoznice za opazovanje prometnega obnašanja.

Struktura dela je taka, da gremo najprej skozi pregled področja simulacij, kjer spoznamo kaj je to simulacija in naštejemo ostale podobne simulatorje. Nato sledi opis standardov in formatov kjer so opisani model IDM in MOBIL in ostali standardi. Za tem pa sledi glavni del naloge, kjer povemo kaj smo dodali in kako.

Poglavje 2

Pregled področja

Izraz simulacija pomeni neke vrste imitacijo oziroma posnemanje operacij procesa realnega sveta ali sistema skozi čas. Dejanje, ki nekaj simulira, najprej zahteva, da je treba razviti model. Ta model predstavlja ključne značilnosti ali vedenja funkcij izbranega, ali abstraktnega sistema, ali procesa. Model predstavlja sistem, v katerem je simulacija operacija sistema v določenem času. Računalniška simulacija je poskus, pri katerem lahko preučujemo model iz realnega življenja in uporabniku prikažemo, kako sistem dejansko deluje. Pri tem spremenljivke simulacije seveda spreminjamo[8].

Simulacija se pojavlja v različnih kontekstih, npr. kot simulacija tehnologije za optimizacijo delovanja, varnostni inženiring, testiranje, treniranje, izobraževanje in video igre. Za študije simulacijskih sistemov pogosto uporabljamo računalniške sisteme. Simulacijo uporabljamo tudi pri znanstvenem modeliranju naravnih ali človeških sistemov, da lahko pridobimo vpogled v njihovo delovanje. Uporabljamo jo tudi za prikaz dejanskih učinkov alternativnih razmer in smeri delovanja. S simulacijo pa si pomagamo tudi takrat, ko ne gre za realni sistem, ker ta ni dostopen, ali pa je lahko nevaren oziroma v njem ne bi bilo mogoče sodelovati, ali pa je bil le oblikovan in še ni zgrajen, ali pa preprosto še ne obstaja.

Ključna vprašanja pri simulaciji vključujejo pridobitev veljavnega vira informacij o ustrezni izbiri ključnih značilnosti in vedenja, uporabo poeno-

stavitev približkov in predpostavk v simulaciji, stopnjo njenega ujemanja s stvarnostjo in veljavnost rezultatov simulacije. Simulacije načeloma delimo na tri vrste:

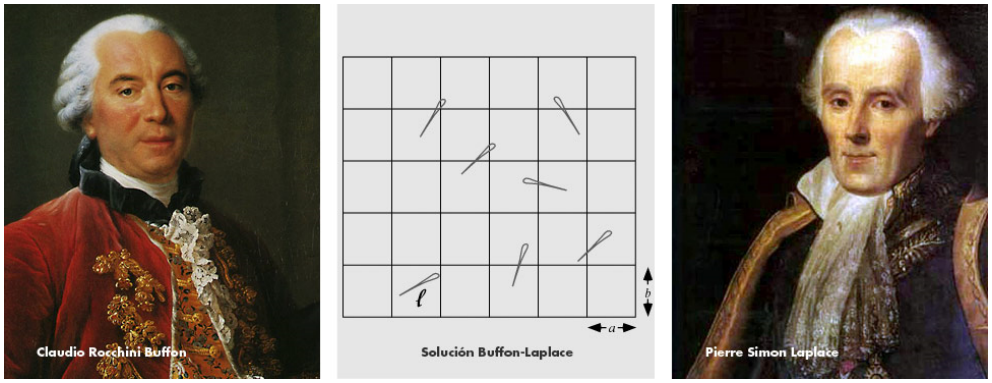
- fizikalna simulacija, pri kateri fizični predmeti nadomeščajo neko pravo stvar. Ti fizični predmeti so pogosto izbrani zato, ker so manjši ali cenejši od dejanskega objekta ali sistema.

- interaktivna simulacija je posebna vrsta fizikalne simulacije, pri kateri človek interaktivno pošilja vhodne podatke simulatorju, posledično pa se v njem spreminja delovanje, odvisno od vhodnih podatkov, ki mu jih človek pošilja. Za primer lahko podamo simulator letenja ali simulator vožnje z avtomobilom.

- simulacija analize okvar se nanaša na simulacijo, s katero lahko ustvarimo okolje ali razmere za ugotavljanje vzrokov okvare opreme. To je najboljši in najhitrejši način, s katerim ugotovimo, kaj povzroča napake na opremi.

Beseda simulacija sega v leto 1777 in se je pojavila z vprašanjem, ki si ga je postavil Buffon ob problemu z iglo [5]. Problem je preprost matematični način, s katerim naj bi dosegli vrednost števila π z zaporednimi poskusi. Temelj tega matematičnega modela je igla, enako dolga ali krajša od razdalje med dvema vzporednima črtama na ravnini. Kolikšna je verjetnost, da bo igla padla na eno do paralelnih in med seboj enako oddaljenih črt?

Leta 1812 je Laplace Buffonovo rešitev izboljšal in popravil, poskus pa poznamo od takrat kot Buffon-Laplaceov. Kasneje je statistik William Sealy Gosset, ki je delal v pivovarni Arthurja Guinnessa (Arthur Guinness Bewery), začel uporabljati statistično znanje v pivovarni in na svojem kmetijskem posestvu. Grosseta so posebej zanimali ječmenovi pridelki, kar ga je privedlo do mnenja, da poskusi ne bi smeli biti namenjeni samo izboljšanju povprečne ravni proizvodnje, ampak tudi razvoju močnejših sevov ječmena, ki jih razlike v zemlji in podnebju ne bi prizadele.



Slika 2.1: Buffon-Laplaceov poskus

Dramatični tehnološki razvoj v zadnjih nekaj desetletjih pomeni, da so nekateri izrazi zdaj del našega vsakdanjega življenja. Zaradi računsko čedalje sposobnejših računalnikov in izboljšavo vizualne tehnike se na tem področju razvijajo nova orodja za usposabljanje, ki temeljijo na inovativnih tehnologijah. Uporaba simulacij se pojavlja kot eden od najučinkovitejših načinov za posredovanje znanja o določenih temah in njihove analize, žal pa so zaradi visoke cene dostopne samo velikim podjetjem ali pa ob znatni pomoči javnega financiranja.

Zahvaljujoč tehnološkemu napredku je simulacija napredovala tako močno, da z njo dosegamo odlično stopnjo zanesljivosti. Zato je mogoče zagnati vrsto matematičnih modelov, ki v kombinaciji z vizualnimi tehnikami natančno odražajo realnost v vsej njeni kompleksnosti.

Poznamo več različnih prometnih simulatorjev, ki so dobro poznani in širše uporabljani[1]:

- SUMO - Simulation of Urban Mobility, verzija 0.10.3
  "**S**imulation of **U**rban **MO**bility" (SUMO) je odprtokodna, dobro prenosljiva, mikroskopska cestno prometna simulacija, oblikovana, da obvladuje velika cestna omrežja.

- Quadstone Paramics Modeller, verzija 6.4.1
  QuadstoneParamics je modularna zbirka mikroskopskih simulacijskih

orodij, ki zagotavljajo močno integrerano platformo za modeliranje celotnega obsega realnega sveta prometnih in prevoznih težav.

- Treiber's Microsimulation of Road Traffic
  Treiber's Microsimulation je osebni pragramski projekt uporabljen v njegovi raziskavi prometnih dinamik in prometnega modeliranja.

- Aimsun, verzija 6.0.4
  Aimsun je simulacijski paket, ki integrira tri tipe prevoznih modelov:orodja prirejanja statičnega prometa;mezoskopski simulator; in mikrosimulator.

- Trafficware SimTraffic, verzija 6
  SimTraffic je del simulacijske aplikacije od TrafficWare Synchro Studio paketa. Trafficware Studiu pomaga kot prometni simulator, ki vključuje tudi aplikacijo za sinhronizacijo semaforjev.

- CORSIM TRAFVU, verzija 6.1
  CORSIM TRAFVU služi za opazovanje prometne simulacije in je del TSIS CORSIM programskega paketa. Nudi animacijo prometnih omrežij.

Izmed naštetih šestih programskih paketov, sta samo dva neplačljiva medtem ko so ostali štirje plačljivi. Plačljive se lahko uporablja za študije le kot demo verzije. Demo verzije lahko uporabljamo le 30 dni zastonj. Še ena značilnost zastonjskih verzij(SUMO and Treiber's Microsimulation of road traffic) je ta, da je njihova izvorna koda dostopna zastonj in si jo lahko za uporabo lahko prenese vsak. V diplomskem delu uporabljamo Treiberjev mikrosimulator cestnega prometa, ker je zastonj in ker je napisan v programskem jeziku Java.

Poglavje 3

Opis standardov in formatov

V tem poglavju so opisani glavni sestavni deli, formati, orodja in sistemi, ki so uporabljeni v našem simulatorju in so potrebni za delovanje simulatorja.

3.1 Vzdolžni model IDM

V tej simulaciji uporabljamo model IDM, ki simulira vzdolžno dinamiko. To so pospeški in zaviranje voznikov glede na ostale udeležence na cesti. Načeloma spremljamo le vozila pred trenutno opazovanim vozilom.

3.1.1 Struktura modela

Model pospeška podrobneje opisuje model inteligentnega voznika(angl. Intelligent Driver Model(IDM))[2] in je namenjen simulaciji vzdolžne dinamike, to je računanju pospeška in zaviranja voznikov. IDM je mikroskopski model prometnega toka. Opisuje stanje prometa v danem trenutku glede na položaj in hitrost vseh simuliranih vozil. IDM je pravzaprav model zasledovanja avta. To pomeni, da je odločitev vsakega voznika, ali bo zaviral ali pospešil, odvisna le od njegove lastne hitrosti in položaja ter položaja in hitrosti vozila tik pred njim, saj morajo vozniki paziti, da se med sabo ne zaletavajo.

Strukturo modela IDM lahko opišemo takole:

- Vplivni dejavniki vhodnega modela so lastna hitrost v, razlika od odbijača do odbijača čelnega vozila in relativna hitrost (razlika hitrosti), delta v (Δv) dveh vozil, ki je pozitivna, ko se vozilo približuje čelnemu vozilu.
- Izhod modela je pospešek $\frac{dv}{dt}$ izbranega vozila za to simulacijo.
- Parametri vozila opisujejo slog vožnje, to je, ali simulirano vozilo vozi počasi ali hitro, previdno ali nepremišljeno itd.

3.1.2 Model enačbe

Model enačbe se glasi takole:

$$\frac{dv}{dt} = a\left[1 - \left(\frac{v}{v_0}\right)^{\delta} - \left(\frac{s^*(v,\Delta v)}{s}\right)^2\right] \tag{3.1}$$

kjer je

$$s^*(v,\Delta v) = s_0 + max\left[0, \left(vT + \frac{v\Delta v}{2\sqrt{ab}}\right)\right] \tag{3.2}$$

Pospešek (3.1) je razdeljen na »želeni« pospešek $\left[1 - (\frac{v}{v_0})^{\delta}\right]$ na prosti cesti in na zaviralne pojemke, povzročene zaradi spredaj vozečih vozil. Pospešek na prosti cesti se zmanjšuje od začetnega pospeška do pospeška nič, ko se hitrost vozila približuje »želeni hitrosti« v_0.

Zavorni izraz temelji na primerjavi »zaželene dinamične razdalje« s^* (3.2) in dejanske razdalje od spredaj vozečega vozila. Če je dejanska razdalja približno enaka s^*, potem zaviranje kompenzira predvsem prosti del pospeška, zato je pospešek skoraj ničen. Poleg tega se s^* poveča dinamično; ko se približuje počasnejšim vozilom, se pospešek zmanjša, ko pa je prednje vozilo hitrejše, pa se pospešek sam poveča do t. i. želene hitrosti. Posledično se pospešek računa na podlagi:

- Zmanjšane razdalje do sprednjega vozila (želimo ohraniti določeno »varnostno razdaljo«).

- Povečane lastne hitrosti (povečanje varnostne razdalje).
- Povečane razlike hitrosti sprednjega vozila (ko se sprednjemu vozilu preveč približuje, lahko nastane nevarna situacija).

3.1.3 Parametri modela IDM

IDM ima intuitivne parametre:

- Želena hitrost vožnje na prosti cesti v_0.
- Želen varnostni čas (razdalja) zasledujočih se vozil T.
- Pospešek v vsakdanjem prometu a.
- »Udoben« zavorni pojemek v vsakdanjem prometu b.
- Najmanjša razdalja od odbijača do odbijača prednjega vozila s_0.
- Pospeškovni eksponent δ.

Na splošno velja, da ima vsaka »enota voznik-vozilo« svoje določene lastnosti oziroma parametre, na primer:

- Za tovornjake so značilne nizke vrednosti v_0 in b.
- Previdni vozniki vozijo z velikim varnostnim časovnim intervalom ali z varnostno razdaljo T.
- Za agresivne voznike so značilna majhna varnostna razdalja T v povezavi z visokimi vrednostmi v_0, a in b.

Pogosto sta za to, da opozorimo na najpomembnejše pojave, dovolj že dve različni vrsti vozil. Standardni parametri, ki se uporabljajo pri simuliranju, so naslednji:

Parameter	Vrednost avtomobila	Vrednost tovornjaka	Opomba
Želena hitrost,v_0	120 km/h	80 km/h	Za mestni promet je dobro omejiti hitrost, medtem ko ostane pri drugih parametrih hitrost,nespremenjena.
Časovni presledek T	1.5 s	1.7 s	Priporočilo slovenskih avtošol je 2 s, vendar se realne vrednosti gibljejo med 2 s in 0,8 s ali celo manj.
Min. razdalja s_0	2.0 m	2.0 m	Razdalja v popolnem mirovanju, npr. ko se pojavijo čakalne vrste, ki jih povzročijo rdeči semaforji.
Pospešek a	0.3 m/s^2	0.3 m/s^2	Realne vrednosti so med 1 in 2 m/s^2
Zaviranje b	3.0 m/s^2	2.0 m/s^2	Realne vrednosti so med 1 in 2 m/s^2

Tabela 3.1: Tabela osnovnih približkov modela

3.1.4 Simulacija modela

Simulacija pomeni številčno integracijo približka rešitev sklopljenih diferencialnih enačb modela. Zato ena definira končni numerični čas posodobitev intervala $\Delta\mathbf{t}$ in se integrira v tem časovnem koraku ob stalnih pospeških. Ta, tako imenovana balistična metoda se glasi:

$$\textbf{nova hitrost: } v(t+\Delta t) = v(t) + (\tfrac{dv}{dt})\Delta t\textbf{,}$$

$$\textbf{nova pozicija: } x(t+\Delta t) = x(t) + v(t)\Delta t + \tfrac{1}{2}(\tfrac{dv}{dt})(\Delta t)^2\textbf{,}$$

$$\textbf{nova razdalja: } s(t+\Delta t) = x_1(t+\Delta t) - x(t+\Delta t) - L_1\textbf{.}$$

Pri čemer je $\frac{dv}{dt}$ IDM pospešek izračunan v času t, x je položaj sprednjega odbijača in L_1 dolžina vodilnega vozila. Za inteligentnega voznika imamo lahko kateri koli časovni korak, manjši od 0.5 sekunde, saj nas privede do enakega rezultata. To pa je dovolj, da se približamo pravi rešitvi.

Strogo gledano je model dobro definiran samo, če imamo vodilno vozilo in če vožnje ne ovira noben drug objekt. Posplošitve pa so zelo enostavne:

- Če nimamo vodilnega vozila ali kakšnega drugega ovirajočega objekta (»free road«), preprosto nastavimo veliko razdaljo, npr. 1000 m.
- Če je naslednik ovirajoči objekt, ki ni vozilo, ampak rdeč semafor, se preprosto postavi nepremično virtualno vozilo, ki se ob preklopu na zeleno luč odstrani.
- Če začne veljati omejitev hitrosti, se zmanjša želena hitrost, če je trenutna vrednost nad to mejo.

3.2 Model menjave pasov MOBIL

Vsak voznik, ki želi priti do želenega cilja in pri tem vozi tudi po večpasovnici, bo slej ko prej moral zamenjati pas. Da bodo naša vozila lahko menjala pas, smo jih morali nekako programirati oziroma jih "naučiti"osnovne logike menjave pasov.

Slika 3.1: Večpasovnica, na kateri je implementiran model MOBIL

Tudi za menjavo pasu je definiran neki standard oziroma model. To je

model menjave pasov MOBIL[3] (ang. Minimizing Overall Braking decelerations Induced by Lane changes). Sprememba voznega pasu se zgodi, če:

- Je potencialni ciljni pas ugodnejši; takrat je izpolnjen spodbujevalni kriterij.

- Je izpolnjen varnostni kriterij.

Na kratko lahko povzamemo oba. Pri prvem se meri lastna pridobljena prednost ob prestopu na ciljni pas v primerjavi z oteževanjem vožnje ali izgubljeno prednostjo, ki jo povzročimo drugim voznikom na ciljnem pasu. Ker imamo različne voznike, tako egoistične kot altruistične, je definiran tudi vljudnostni faktor, ki se meri v kombinaciji z izgubo prednosti, ki smo jo vzeli voznikom na ciljnem pasu. Število je največkrat manjše od 1. Varnostni kriterij je izpolnjen, če se vozilo na ciljnem pasu lahko ustavi tako, da bo med voziloma še vedno neka varnostna razdalja, ki je že prej definirana.

Parameter	Tipična vrednost	Opombe
Vljudnostni faktor p	0 ... 0.5	Podrobnosti glejte spodaj
Največja varna upočasnitev b_{save}	4 m/s^2	Mora biti manjša od maksimuma upočasnitve, ki se giblje okoli 9 m/s^2
Prag a_{thr}	0.2 m/s^2	Mora biti pod najnižjo sposobnostjo pospeševanja (IDM-parameter a) pri, vsakem tipu vozila
Usmerjanje na desni vozni pas Δb	0.2 m/s^2	Samo za evropska pravila

Tabela 3.2: Tabela osnovnih nastavitev modela MOBIL

3.2.1 Model menjave voznega pasu MOBIL ima svoje značilnosti

Medtem ko drugi modeli menjave voznega pasu navadno predvidevajo le egoistično vedenje voznikov, torej $p = 0$, upoštevamo pri našem modelu različno vedenje s spreminjajočim se faktorjem p, imenovanim vedenjski faktor.

- $p > 1$ => zelo nesebično vedenje.
- p med 0 in 0,5 => realistično obnašanje. Drugi vozniki imajo manjšo prednost, vendar se je ne zanemarja. Ta funkcija pomeni, da imamo lahko voznike, ki se vedejo različno: tako egoistično kot altruistično. To je zajeto v modelu MOBIL.
- $p = 0$ => zgolj sebično vedenje; vendar tudi sebični vozniki ne prezrejo varnostnega kriterija.
- $p < 0$ => zlonamerna osebnost, ki rada ovira druge voznike, tudi za ceno lastne ogroženosti.

Poseben primer predstavljata parametra $p = 1$ in $a_{thr} = 0$. Pri tem se pas menja, kadar se vsota pospeškov vseh prizadetih voznikov povečuje po spremembah, ali ekvivalentno, ko skupne pojemke zmanjšamo. Ta učinek je tudi ponudil kratico za ta model.

3.3 Zavezujoča javanska arhitektura XML(JAXB)

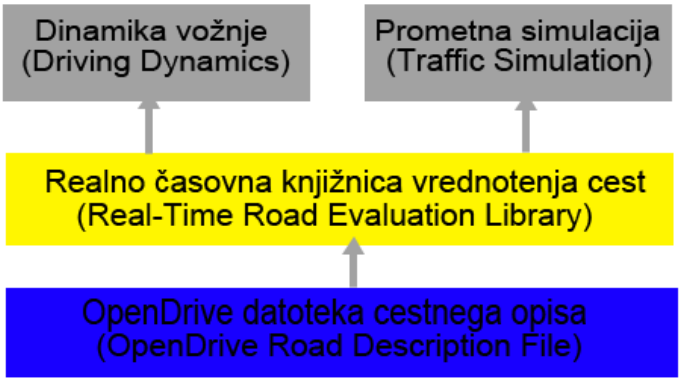
Javanska arhitektura z XML povezavo (ang. Java arhitecture for XML binding (JAXB)) nam omogoča, da v datoteki definiramo ali opišemo postavitev sheme, ki jo kasneje lahko uporabljamo. Na podlagi tega opisa z orodjem Ant generiramo razrede, ki jih potrebujemo za objektni prenos podatkov iz jezika XML v Javo. Shema, ki jo sestavimo, nam pove, po kakšni strukturi in zaporedju moramo definirati elemente.

3.4 Format OpenDrive

OpenDRIVE je odprtokodni format, ki v datotekah predstavlja logični opis cestnih omrežij[7]. Ob izdatni podpori simulacijske industrije ga je razvila ekipa simulacijskih strokovnjakov, ki ga tudi vzdržuje. Projekt OpenDRIVE se je začel pred nekaj leti, ko je začelo podjetje VIRES graditi podatkovne zbirke različnih simulatorjev vožnje, ki naj bi z vmesnikom podatkovnih baz (npr. z vozno dinamiko ali avtonomijo prometa) dodalo opis logike cestnega omrežja za vsako bazo podatkov.
Datotečni format OpenDRIVE zagotavlja naslednje:

- Format XML.
- Hierarhično strukturo.
- Analitično definicijo cestne geometrije (detektorski elementi, nadmorska višina, širina pasu itd.).
- Različne vrste pasov.
- Vključene prednostne naloge križišč.
- Logične povezave med pasovi.
- Vključene odvisnosti znakov in signalov.
- Krmilnike signalov(npr. v križiščih).
- Površinske lastnosti cest.
- Objekte cest in stranskih cest.
- itd.

Datoteke OpenDRIVE so namenjene opisu celotne cestne mreže z vsemi podatki, ki sodijo v cestno okolje. Naslednja slika prikazuje tipično vključevanje datoteke OpenDRIVE v simulacijo.



Slika 3.2: Struktura uporabe datoteke OpenDRIVE

3.5 Razvojno okolje Netbeans

NetBeans je integrirano razvojno okolje (IDE), ki naj bi se sprva razvijalo s programskim jezikom Java. Uporablja se lahko tudi z drugimi jeziki, zlasti s PHP, C/C++ in HTML5. Hkrati je aplikacija tudi okvirna platforma za namizne aplikacije Jave in druge. Platforma NetBeans nam dovoljuje razvoj aplikacij v množici modularnih programskih komponent, imenovanih moduli. Aplikacije na podlagi platforme NetBeans platforme lahko razširijo tudi tretjeosebni razvijalci, kar pripomore k večji prenosljivosti programov. Skupina NetBeans aktivno podpira in nadgrajuje izdelek iz povratnih informacij in odziva širše javnosti.

3.6 Sistem Maven

Maven je velik Java projekt izdelave in orodje za upravljanje iz Apache programske skupnosti. Eden glavnih povdarkov sistema Maven je orodje za gradnjo, ki opravlja podobne naloge kot Ant, čeprav je več kot Ant.

Zelo pomemben vidik sistema Maven je uporaba odlagališča za upravljanje jar datotek v različnih projektih. V preprostem programskem razvojnem okolju, lahko delamo na projektu in preverimo jar datoteke neposredno v

našem projektu v sistemu za nadzor različic in to lahko naredimo v vsakem projektu na katerem delamo. Ta sistem deluje dobro v razmeroma preprostih situacijah, vendar postane nerodno, ko projekti postajajo kompleksnejši in večji v številu razvijalcev. V sistemu Maven so jar datoteke shranjene v oddaljenih odlagališčih (ang. remote repositories) in jih prenesemo na svoj lokalni računalnik v lokalno odlagališče (ang. local repository), kolikor jih potrebujemo. Značilno je, da so te iste jar datoteke dostopne preko projektov. Maven si naredi zelo enostavno za upravljanje različnih verzij jar datotek in združi sklope povezanih jar datotek. Maven lahko upravlja naloge kot je ustvarjanje koristnih dokumentacij o projektu.

Poglavje 4

Simulacija prometnih tokov

V tem poglavju opišemo simulator in dodatke, ki smo jih implementirali v naš simulator. To so gradnja osnovnega semaforiziranega križišča, opis kritičnih delov cest(zakaj so potrebni), delno cestno omrežje kjer smo sestavili križišče iz realnega sveta, grafična vmesnika za sestavo cestnih omrežij in gradnjo križišč.

Diplomska naloga se sklicuje na simulacijo prometnih tokov na območju Ljubljane. Simulator pa smo nadgradili splošneje, tako da ga je mogoče uporabljati tudi za druga mesta, saj uporabnik lahko sestavi popolnoma novo omrežje, ki ga zanima.

Namestili smo potrebno programsko okolje in pakete, tako imenovani NetBeans in razvojni komplet Java (ang. Java Development Kit), ki je nujen za delovanje. Razvojni komplet JDK je zbirka programskih orodij in vsebuje glavne komponente, vključno s prevajalnikom in drugimi dodatnimi orodji, ki se ob uporabi izkažejo za koristna. Ko smo imeli nameščena vsa potrebna orodja, smo začeli raziskovati področje simulacije prometa. Sprva smo imeli kodo, ki je opisovala le določene primere, kot so krožišče, cestna zapora, vključitev ceste v promet, omejitev hitrosti in način obnašanja avtomobilov med vožnjo na klancu. Ker pa smo našli naprednejšo verzijo simulatorja smo to tudi uporabili. Najprej smo kreirali enopasovno cesto z določenim vhodom vozil na sekundo, hkrati pa izhodni del, kjer avtomobil izgine. To je

bilo videti takole:

Slika 4.1: Enopasovnica s tokom vozil

Vozila se premikajo z leve proti desni, in sicer z nastavitvijo vhodnega toka 1000 vozil na uro. Ker se vozila lahko premikajo z različnimi hitrostmi, morajo vsebovati neko logiko ali model.

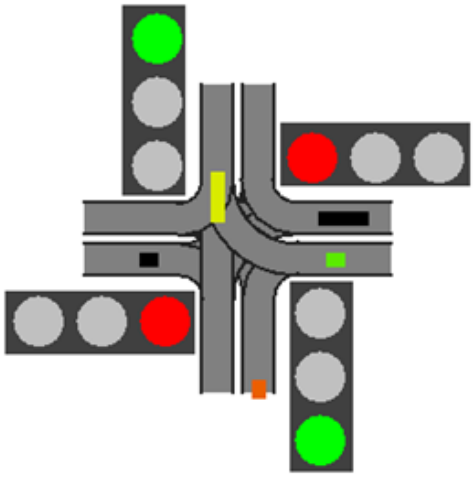
4.1 Nadgradnja simulatorja

Program uporablja Maven, zgrajen je torej po modulih, ti pa se delijo na:

- Jedro, ki skrbi za izračun potrebnih podatkov (računanje položajev vozil, hitrosti itd.).
- Izrisovalni modul, ki glede na podatke, ki jih izračuna jedro, izriše vse potrebne elemente in jih prikaže uporabniku.
- Branje datotek, pri čemer je definirana scena ali postavitev cestišča. Na podlagi podatkov o sceni se generirajo ustrezni objekti. Scena je definirana v XML.

V program smo dodali še funkcionalnost, ki omogoča preskok iz enega na več cestnih segmentov, kar prej ni bilo mogoče. To je bilo treba narediti zato, da smo lahko zgradili osnovno križišče. Torej križišče s štirimi vhodnimi in štirimi izhodnimi cestnimi segmenti, ki imajo vsak po en pas. Ker nam je izvorna koda dopuščala le, da ima vsak pas ceste samo en izvorni in en ponorni segment, smo to spremenili. Če pomislimo, da se peljemo v tako križišče, imamo tri možnosti. Lahko zavijemo levo, desno ali pa gremo naravnost. Enako velja za izhodne segmente. Vsak izhodni segment ima lahko hkrati več izvorov, saj lahko na izhodno cesto pripeljemo iz treh smeri. Zato

je bila ta sprememba nujna. Šele po njej smo lahko začeli gradnjo križišča. V označevalnem jeziku smo definirali vhodne in izhodne ceste križišča, v njem pa 12 segmentov, ki bodo izhode in vhode povezovali.



Slika 4.2: Osnovno križišče

Definirati smo morali tudi izvorne ceste. To so ceste, na katerih se vozila generirajo in se postavijo na cesto, ko začnejo vožnjo. Dodali smo tudi semaforje in dolžino trajanja prižganih luči na njem. Problem pa je nastal, ko je vozilo pripeljalo do razcepa, v katerem so na voljo trije ponori: levo, desno in naravnost. Vozilo ni vedelo, na kateri ciljni segment naj se postavi, zato je prihajalo do napak. Zato smo morali, preden bi spustili promet na cesto, za vsak izvorni segment ročno vnesti zaporedje cestnih segmentov vseh mogočih poti. Vsakemu vozilu smo ob generiranju dodelili eno izmed naštetih poti, po kateri lahko pride do končnega segmenta. To pa je bila le začasna rešitev, saj bi bilo pri večjem omrežju ročno vnašanje vseh mogočih poti nesmiselno in zamudno. Zato smo na podlagi cest in njihovih povezav izdelali graf, na katerem so cestni segmenti predstavljali povezave, točke pa povezave med njimi (izvore in ponore vsakega segmenta). Graf smo zgradili s pomočjo javanske knjižnice JGraphT. To je brezplačna knjižnica, ki nam ponuja matematične objekte grafov in algoritme, ki jih v njih lahko izvajamo. Podpira povezane grafe, nepovezane, utežene, neutežene, psevdografe itd.. Za iskanje poti smo

uporabili algoritem Dijsktra, ki mu določimo začetno in končno točko, on pa nam ponudi najugodnejšo pot.

4.1.1 Kritični deli cest

Najprej moramo vedeti, zakaj smo morali implementirati kritične dele cest. Ker vsebuje naš program segmente, ki se lahko tudi prekrivajo, bi bilo dobro, da avtomobili ne bi vozili drug čez drugega. Lep primer tega je slika osnovnega križišča, ko avtomobili zavijajo levo. Avtomobil ne sme zaviti levo, dokler cesta ni prazna. To smo naredili tako, da smo definirali kritične dele cest. Kritični del ceste je definiran kot:

- $cesta_1$, ki predstavlja problematičen del, na katerem se segmenta lahko sekata;
- blokada na $cesti_2$: blokada je postavljena, medtem ko je $cesta_1$ zasedena oziroma je na njej neko vozilo.

Definiramo lahko tudi sprejemljive hitrosti vozil na kritičnih delih cestišč. Nasproti vozeče vozilo pelje na primer zelo počasi; potem blokade na zavoju v levo ni in avtomobil lahko zavije levo.

4.1.2 Delno cestno omrežje

Naslednji cilj, ki smo si ga zadali, je bila postavitev omrežja na območju ljubljanske obvoznice. S sestavljanjem cest smo začeli pri najbolj kompleksnih križiščih(Slika 4.3). Da bi križišča postavili natančneje, smo si za ozadje nastavili sliko zemljevida, kar nam je vse skupaj nekoliko olajšalo. Ker pa z zemljevida ni bilo mogoče razbrati, koliko pasov ima neki vhod v križišče, smo si pomagali s storitvijo ulični pogled (ang. Street View) podjetja Google. Sprva smo sestavili križišče, v katerem se sekata Tivolska in Celovška cesta. Vsak cestni segment je definiran s svojim unikatnim številom *id*.

Ker smo se lotili kompleksnejše mreže, se je bilo treba dogovoriti o dodelitvi id-številk. Vsako križišče ima vhodne in izhodne ceste in ceste znotraj

Slika 4.3: Primer uporabe na konkretnih primerih

križišča. Poleg teh dveh klasifikatorjev imamo tudi ceste, ki križišča med seboj povezujejo. Za vse tri smo rezervirali intervale *id*-številk, to pa zato, da bi jih laže ločili med seboj. Ko smo omenjeno križišče sestavili, večjih težav nismo imeli. Težave so nastopile, ko smo dodali še drugi dve križišči in vse tri povezali med sabo.

Ko smo v omrežje spustili večje število vozil, so se ta začela zaletavati. V trenutku, ko sta vozili trčili, je simulator začel delovati počasneje. To pa zato, ker je relaksacija vozil računsko potratna. Relaksacija je izračun tega, kateremu vozilu bomo dali prednost pri nadaljevanju poti. Večina trkov se je zgodila ob prehodu vozil iz segmentov, ki imajo isti ponorni/ciljni segment. Ko sta na primer dve vozili v istem trenutku vstopili v isti segment, sta imeli enake položaje, ob tem pa je program prepoznal trk. Iz tega smo izluščili, da moramo v bližnji prihodnosti segmentom dodati neke vrste opazovalce. Vozilo, ki je na segmentu z opazovalcem, začne opazovati del ceste, ki smo jo predhodno definirali kot opazovano. Če na tej cesti ni vozila, lahko vozilo, ki je na cesti z opazovalcem, vožnjo nadaljuje, sicer se ustavi. Enaka težava se pojavi pri enostavnih križiščih, v katerih vozila zavijajo levo. Če opazovalcev nimamo, bo zagotovo prišlo do bočnega trčenja. Problem smo rešili z že predhodno omenjenimi kritičnimi deli cestnih segmentov. Vseeno pa smo opazili še eno napako. Kot smo že omenili, se vsakemu vozilu ob vstopu dodeli

pot. Izkaže se, da vsako vozilo ne more nadaljevati poti na segment, ki je v njegovi poti določen kot naslednik trenutnega. To se zgodi največkrat zaradi nezmožnosti menjave pasov. Če vozilo pasu ne more zamenjati, nadaljuje pot na segment, ki je definiran kot naslednik trenutnega pasu. Potem je vozilo izgubljeno, saj je skrenilo s poti, ki mu je bila dodeljena. Sprva smo to rešili tako, da smo vozila, ki so se izgubila, preprosto izločili iz prometa. Nato pa smo problem rešili tako, da se je za izgubljena vozila ponovno izračunala pot do nekega naključnega izhodnega segmenta, ki ga iz trenutnega cestnega segmenta vozilo lahko doseže. Seveda pa smo se pogovorili tudi o tem, kako bomo kasneje ta problem rešili. Doseči bi morali, da bi vozilo vedno zapeljalo na pas, ki je zanj najugodnejši. Če bi hoteli doseči to, bi morali vsakemu voznemu pasu določiti tip, ki bi nam tako kot smernice na cesti povedal, kater pas je za nas ugodnejši, po katerem lahko nadaljujemo pot, ne da bi zašli z nje. Tako bi lahko odpravili problem izgube vozil, hkrati pa bi se bolj približali realnemu toku prometa.

Slika 4.4: Omrežje večjih cest znotraj ljubljanske obvoznice (levo) in prikaz prometnega pretoka (desno)

Da bi si vse skupaj nekoliko poenostavili, smo predpostavili, da naj bodo vsa križišča osnovna. Dodali smo funkcijo, ki ujame vsak klik miške. Ob vsakem kliku pa nam v tekstovno datoteko shrani koordinate položaja miške, kjer se je klik zgodil. Ko smo imeli položaje križišč, smo izdelali funkcijo, ki nam je na podlagi teh podatkov v označevalnem jeziku XML ustvarila

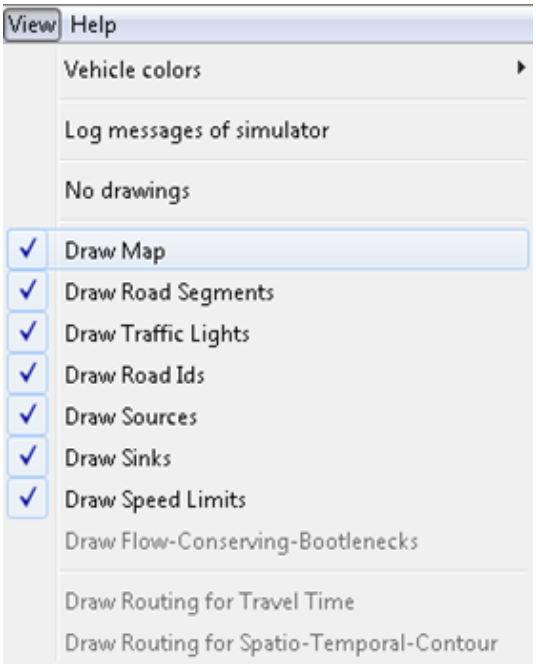
kodo vseh cestnih segmentov za vsa križišča. Tako smo dobili nepovezana križišča. Ta so potrebovala le še medsebojne povezave, za katere pa smo potrebovali identifikacijska števila cest, ki jih cesta povezuje. Tudi tu smo koordinate pridobili s pomočjo funkcije za zajem koordinat. Ker pa se ceste v shemi OpenDrive določa s koordinatami začetka, njihovo dolžino in kotom, pod katerim ležijo, smo iz zajetih koordinat preračunali potrebne podatke in iz njih ustvarili kodo postavitve cestišča. Tako smo prihranili veliko časa, hkrati pa ugotovili, da moramo naš program bolj približati uporabniškemu vmesniku za lažjo in hitrejšo sestavo cestnega omrežja.

4.1.3 Grafični vmesnik za sestavo cestnih omrežij

Za lažjo sestavo cestnih omrežij smo pripravili grafični vmesnik, zato da nam posameznih segmentov ne bo treba ročno vnašati v formatu XML OpenDrive. Dodali smo tudi možnost izbire(Slika 4.5), kaj naj se izriše na naš Canvas. Canvas je objekt, na katerem izrisujemo vse v zvezi s simulacijo (slika ozadja, cestni segmenti, semaforji, id cest, izvori in ponori vozil in omejitve hitrosti).

Če preskočimo na glavni del grafičnega vmesnika, vidimo, da smo dodali v našo orodno vrstico tudi možnost Edit, kjer lahko izberemo način urejanja(Slika 4.6). Ko smo v načinu urejanja, se simulacija ustavi, saj se takrat lahko kreirajo novi segmenti in križišča. Takrat se ustvari tudi navidezni obroč okoli segmentov, tako da lahko izbiramo, kateri segment bomo urejali. Zato da bi novo zgrajeno omrežje lahko shranili, smo dodali tudi funkcionalnost Save as XML, pri čemer se vsa novo generirana cestišča in njihove spremembe, povezave shranijo v formatu XML OpenDRIVE.

Ko dodamo ravno ali krivuljasto cesto, se nam prikažejo tudi pomembni podatki o novo zgrajeni cesti (koordinati x in y, dolžina segmenta in kot, pod katerim cesta leži; če je cesta krivuljasta, pa se nam prikažeta tudi radij in dolžina loka, ki ga želimo ustvariti). Če nam cesta ustreza, pritisnemo gumb Apply. Takrat se cesta doda v obstoječe omrežje, ni pa še shranjena v strukturi XML. Če hočemo, da se cesta v omrežju ohrani, moramo shraniti tudi spremembe v datoteki XML, kar nam omogoča JAXB.



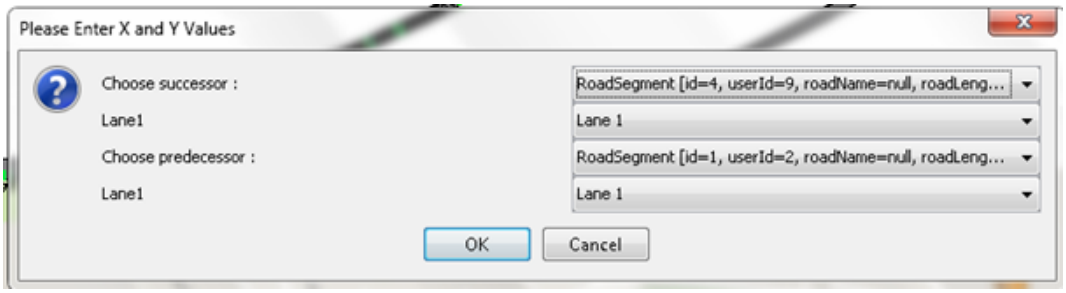
Slika 4.5: Opcije izbire izrisa

Slika 4.6: Urejevalna vrstica

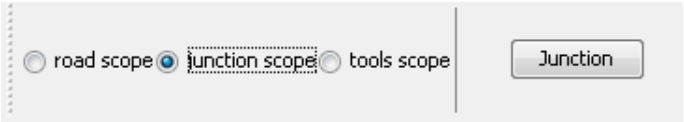
Ceste lahko tudi dodajamo in medsebojno povezujemo. Med sabo morajo biti povezane zato, da avtomobili vedo, kateri je njihov naslednji ali predhodni cestni segment. Pri povezovanju cest moramo cesti določiti dvoje. To sta ponorni in izvorni pas katere koli ceste, ki jo izberemo, da bi jo povezali(Slika 4.7).

4.1.4 Dodajanje križišč

Poleg vseh naštetih funkcij smo dodali tudi možnost dodajanja križišč po meri. To najdemo pod rubriko Edit ob kliku na radio gumb junction scope(Slika 4.8). Odpre se nam možnost Junction.

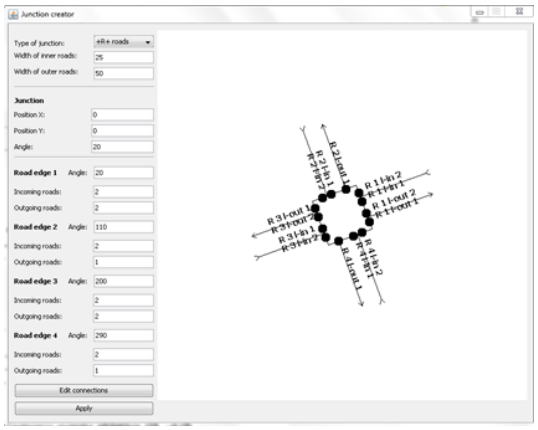


Slika 4.7: Povezavni dialog JPanel



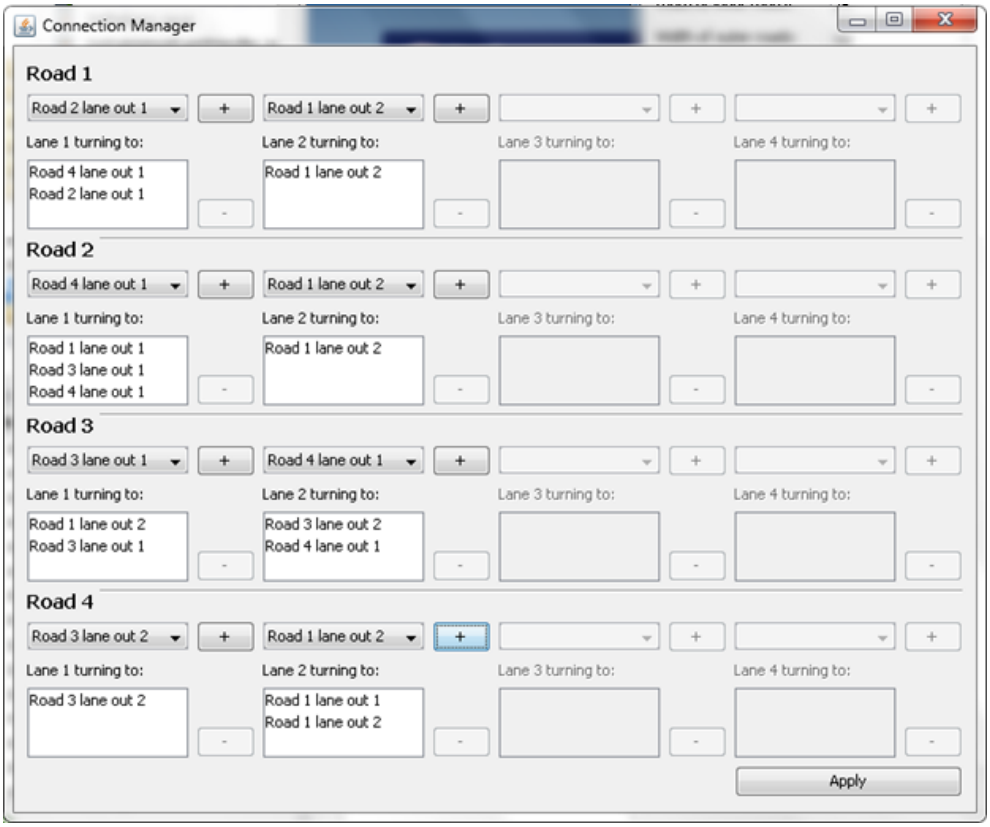
Slika 4.8: Orodna vrstica za urejanje križišč

Ko izberemo to možnost, se nam odpre okno za definicijo križišča(Slika 4.9). Definiramo tip križišča(T- ali Y- ali +-križišče), širino cest znotraj križišča, dolžino zunanjih cest (vhodnih in izhodnih), položaj (x in y), kot, pod katerim leži, na koncu pa definiramo še, koliko vhodnih in izhodnih cest ima vsak rob ceste. Če je torej križišče tipa +, imamo 4 robove, za katere moramo poznati število vhodnih in izhodnih cest v križišče.



Slika 4.9: Urejevalnik križišč

Ker so znotraj križišča tudi povezave, jih moramo definirati, sicer bi ostale



Slika 4.10: Urejevalnik povezav znotraj križišča

nepovezane in bi križišče predstavljalo samo ceste, nametane druga na drugo, kar pa ni naš cilj. Ko izpolnimo vse podatke o križišču, ki ga želimo, se pojavi opcija Edit Connections, pri čemer se ob kliku odpre nov JFrame (Slika 4.10), kjer nastavimo, katere vhodne ceste so povezane s katerimi izhodnimi. Predpostavili smo, da ima vsak rob križišča največ štiri vhodne ceste; za vsako vhodno cesto moramo našteti izhodne ceste, s katerimi je povezana.

4.2 Logični opis cestnega omrežja

OpenDRIVE je odprtokodna specifikacija za logični opis cestnega omrežja. Po specifikaciji OpenDRIVE torej naštejemo ceste, dodamo njihove dolžine ter definiramo njihove predhodnike in naslednike. Ker smo v kodi že imeli način definicije objektov(JAXB), ni bilo potrebe po tem, da bi uporabili ogrodje Spring.

Shemo OpenDrive smo nekoliko spremenili in dodali možnost, da se posamezna križišča lahko kreira kot objekte. Definicija sheme je približno taka:

```
<xsd:element name="junctionRoads">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="road" minOccurs="1" maxOccurs="unbounded"/>
      <xsd:attribute name="id" type="xsd:string"/>
      <xsd:attribute name="trafficLightsStartTime" type="xsd:int" default="0"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Kot smo dejali, smo dodali tudi funkcionalnost dodajanja kritičnih intervalov, zato smo v značko z imenom junctionRoads dodali tudi to funkcionalnost:

```
<xsd:element name="criticalIntervals" minOccurs="0" maxOccurs="1">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="interval" minOccurs="0" maxOccurs="unbounded">
        <xsd:complexType>
                  <xsd:attribute name="roadSegmentId" type="xsd:int" />
                  <xsd:attribute name="lane" type="xsd:int" />
                  <xsd:attribute name="startPosition" type="xsd:double" />
                  <xsd:attribute name="endPosition" type="xsd:double" />
                  <xsd:attribute name="blockadeRoadSegmentId" type="xsd:int" />
                  <xsd:attribute name="blockadeLane" type="xsd:int" />
                  <xsd:attribute name="blockadePosition" type="xsd:double" />
          <xsd:attribute name="type" type="xsd:string" />
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Vendar je to le definicija sheme, iz katere se nato generirajo objekti določenih tipov (junctionRoads, criticalIntervals). Primer uporabe in de-

finicije neke ceste pa je predstavljen takole:

```
<road name="R19" length="19.5" id="19" junction="-1">
  <link>
    <predecessor elementType="road" elementId="3" contactPoint="end"/>
    <successor elementType="road" elementId="4" contactPoint="end"/>
  </link>
  <planView>
    <geometry s="0.0" x="1454" y="1111" hdg="-1.5708" length="19.5">
      <arc curvature="-0.08"/>
    </geometry>
  </planView>
  <lanes>
    <laneSection s="0.0">
      <right>
        <lane id="-1" type="driving" level="0">
          <link>
            <predecessor id="-1"/>
            <successor id="-1"/>
          </link>
          <width sOffset="0.0" a="10.0" b="0.0" c="0.0" d="0.0"/>
        </lane>
      </right>
    </laneSection>
  </lanes>
</road>
<criticalIntervals>
  <interval roadSegmentId="0" lane="1" startPosition="0.0" endPosition="30.0"
  blockadeRoadSegmentId="15" blockadeLane="1" blockadePosition="19.625"/>
</criticalIntervals>
```

To pa so le poenostavljeni primeri. Za shranjevanje na novo dodanih objektov smo bili prisiljeni dodati funkcionalnost samododeljevanja cestnih identifikatorjev. Za Save as XML smo ustvarili funkcijo, ki nam shrani vse spremembe, ki smo jih dodali s pomočjo našega grafičnega vmesnika, vse, kar smo spremenili ali izbrisali. Vse se shrani v obliki XML (podobno kot v zgornjem primeru).

Poglavje 5

Sklepne ugotovitve

Ustvarili smo zelo uporaben simulator prometnih tokov. Simulator deluje dobro, saj pokaže kje nastajajo zastoji, kar nam da vedeti, kje moramo spremeniti intervale semaforjev. V diplomski nalogi smo ugotovili, da se problem lahko deli na neskončno mnogo podproblemov. Zato smo stvari nekoliko poenostavili. Priprava programa za vzporedni zagon nam ni povzročala nobenih težav. Za iskanje najbolj optimalne rešitve smo pripravili vrsto funkcij, ki so nam vračale števila, rezultat simulacije. Ob vzporednem zagonu z različnimi nastavitvami semaforjev smo dobili več različnih rezultatov, probali pa smo dobiti čim boljšega. Ker je omrežje predstavljalo poenostavljeno omrežje z osnovnimi križišči, žal nismo prišli do realnih rezultatov, saj simulator še ni bil razvit do te mere, da bi lahko sestavili realnejše omrežje. Hkrati smo predpostavili, da je omejitev hitrosti povsod enaka 50 km/h, kar je standard za ceste v naseljih. V simulaciji se razločno vidi, kje prihaja do zastojev, zato lahko sklepamo, da moramo tam zamenjati intervale semaforjev, če hočemo, da bo promet tekel hitreje. Simulator omogoča gradnjo poljubno velikih mest, s tem pa moramo vzeti v zakup dejstvo, da je pri večjih omrežjih tudi več avtomobilov in posledično več preračunavanja zato simulator začne tudi počasneje delovati. Zaradi tega bi v poštev prišla kakšna OpenCL ali CUDA paralelizacija računanja hitrosti/premikanja vozil. Mesto v velikosti Ljubljane deluje normalno brez večjih zakasnitev, tako da ocenjujem, da bi

se dalo sestaviti tudi nekoliko večje mesto od Ljubljane in bi zadeva še vedno delovala zadovoljivo.

Naloga je zelo uporabna za večja mesta s semaforizirano ureditvijo cestišč. Za vsako mesto se namreč lahko sestavi omrežje, ki ga nato lahko analiziramo, in pridobi čim boljše intervale za menjavo luči na semaforjih. Izboljšava, ki bi jo še dodali, je samodejno generiranje omrežij, tako da ne bi uporabljali le osnovnih križišč, ampak realna križišča v realnem omrežju. Nad tem omrežjem bi se izvajala optimizacija delovanja semaforjev, ki bi realno ocenila intervale semaforjev.

Literatura

[1] G. Kotusevski and K.A. Hawick, *A Review of Traffic Simulation Software*, Computer Science, Institute of Information Mathematical Sciences, Massey University at Albany, Auckland, New Zealand.

[2] Martin Treiber, Arne Kesting *Traffic Flow Dynamics* Data, Models and Simulation. (2013)

[3] M. Treiber and D. Helbing, *Realistische Mikrosimulation von Straßenverkehr mit einem einfachen Modell* , 16. Symposium "Simulationstechnik ASIM 2002" Rostock, 10.09 -13.09.2002, edited by Djamshid Tavangarian and Rolf Grützner pp. 514–520.

[4] Jason van Zyl, Brian Fox, John Casey, Bruce Snyder, Tim O'Brien, Eric Redmond *"Maven: The Definitive Guide"*, Published: O'Reilly (Edition 1: October 1, 2008).

[5] Lander simulation (2014). *History of simulation* [Online]. Dosegljivo: `http://www.landersimulation.com/eng/training-with-simulation/the-world-in-motion/history-of-simulation/`. [Dostopano 12. 3. 2015].

[6] MovSim. [Online]. Dosegljivo: `http://www.movsim.org/#about`. [Dostopano 12. 3. 2015].

[7] OpenDrive specification. [Online]. Dosegljivo: `http://www.opendrive.org/project.html`. [Dostopano 12. 3. 2015].

[8] Simulation. *Wikipedia* [Online]. Dosegljivo: `http://en.wikipedia.org/wiki/Simulation`.[Dostopano 12. 3. 2015].